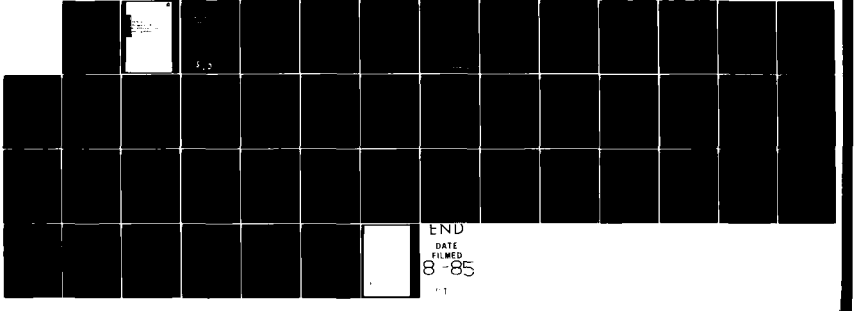
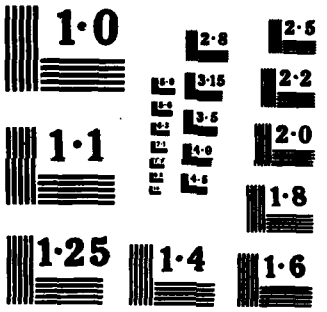
AD-A156 647 METHODS FOR DESIGNING SOFTWARE TO FIT HUMAN NEEDS AND 1/1
CAPABILITIES: PROCE..(U) NATIONAL RESEARCH COUNCIL
WASHINGTON DC COMMITTEE ON HUMAN FA..
UNCLASSIFIED N S ANDERSON ET AL. MAY 85 N00014-85-G-0093 F/G 9/2 NL



END
DATE FILMED
8-85
DTIC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6

NAS
NAE
IOM
NRC

Methods for Designing Software to Fit Human Needs and Capabilities

Proceedings of the Workshop on Software Human Factors

Methods for Designing Software to Fit Human Needs and Capabilities

Proceedings of the Workshop on Software Human Factors

Nancy S. Anderson and Judith Reitman Olson, Editors

Committee on Human Factors
Commission on Behavioral and Social Sciences and Education
National Research Council

NATIONAL ACADEMY PRESS
Washington, D.C. 1985

NOTICE: The project that is the subject of this report was approved by the Governing Board of the National Research Council, whose members are drawn from the councils of the National Academy of Sciences, the National Academy of Engineering, and the Institute of Medicine. The members of the committee responsible for the report were chosen for their special competences and with regard for appropriate balance.

This report has been reviewed by a group other than the authors according to procedures approved by a Report Review Committee consisting of members of the National Academy of Sciences, the National Academy of Engineering, and the Institute of Medicine.

The National Research Council was established by the National Academy of Sciences in 1916 to associate the broad community of science and technology with the Academy's purposes of furthering knowledge and of advising the federal government. The Council operates in accordance with general policies determined by the Academy under the authority of its congressional charter of 1863, which establishes the Academy as a private, nonprofit, self-governing membership corporation. The Council has become the principal operating agency of both the National Academy of Sciences and the National Academy of Engineering in the conduct of their services to the government, the public, and the scientific and engineering communities. It is administered jointly by both Academies and the Institute of Medicine. The National Academy of Engineering and the Institute of Medicine were established in 1964 and 1970, respectively, under the charter of the National Academy of Sciences.

The Committee on Human Factors in the Commission on Behavioral and Social Sciences and Education ~~is~~ sponsored jointly by the Air Force Office of Scientific Research, the Army Research Institute for the Behavioral and Social Sciences, the Office of Naval Research, the National Aeronautics and Space Administration, and the National Science Foundation.

This work relates to Department of the Navy Grant No. N00014-85-G-0093 issued by the Office of Naval Research under Contract Authority NR 196-167. However, the content does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

Available from: Committee on Human Factors, National Research Council, 2101 Constitution Avenue, N.W., Washington, D.C., 20418.

WORKSHOP ON SOFTWARE HUMAN FACTORS

NANCY S. ANDERSON (Chair), Department of Psychology, University of Maryland
ELIZABETH K. BAILEY, Software Metrics, Inc., Falls Church, Va.
STUART L. CARD, Xerox Palo Alto Research Center, Palo Alto, Calif.
JOHN M. CARROLL, Watson Research Center, IBM Corporation, Yorktown Heights, N.Y.
ALPHONSE CHAPANIS, Industrial and Human Factors Consulting Services, Baltimore, Md.
H. REX HARTSON, Department of Computer Science, Virginia Polytechnic Institute and State University
DAVID R. LENOROVITZ, Computer Technology Associates, Inc, Englewood, Colo.
MARILYN M. MANTEI, Graduate School of Business Administration, University of Michigan
JUDITH REITMAN OLSON, Graduate School of Business Administration, University of Michigan
RICHARD W. PEW, Bolt Beranek and Newman Laboratories, Inc., Cambridge, Mass.
PHYLLIS REISNER, IBM Research, San Jose, Calif.
JANET WALKER, Symbolics, Inc., Cambridge, Mass.
JOHN A. WHITESIDE, Digital Equipment Corporation, Maynard, Mass.
ROBERT C. WILLIGES, Department of Industrial Engineering and Operations Research, Virginia Polytechnic Institute and State University

COMMITTEE ON HUMAN FACTORS

THOMAS B. SHERIDAN (Chair), Mechanical Engineering and Applied Psychology, Massachusetts Institute of Technology
NANCY S. ANDERSON, Department of Psychology, University of Maryland
ALPHONSE CHAPANIS, Industrial and Human Factors Consulting Services, Baltimore, Md.
JEROME ELKIND, Systems Development, Xerox Corporation, Palo Alto, Calif.
BARUCH FISCHHOFF, Decision Research (a branch of Perceptronics, Inc.), Eugene, Ore.
OSCAR GRUSKY, Department of Sociology, University of California, Los Angeles
ROBERT M. GUION, Department of Psychology, Bowling Green State University
JULIAN HOCHBERG, Department of Psychology, Columbia University
K.H. EBERHARD KROEMER, Ergonomics Laboratory, Virginia Polytechnic Institute and State University
THOMAS K. LANDAUER, Bell Communications Research, Morristown, N.J.
JUDITH REITMAN OLSON, Graduate School of Business Administration, University of Michigan
RICHARD M. PEW, Bolt Beranek and Newman Laboratories, Inc., Cambridge, Mass.
STOVER H. SNOOK, Ergonomics Laboratory, Liberty Mutual Research Center, Hopkinton, Mass.
ROBERT C. WILLIGES, Department of Industrial Engineering and Operations Research, Virginia Polytechnic Institute and State University

STANLEY DEUTSCH, Study Director
ANNE M. SPRAGUE, Administrative Secretary

CONTENTS

FOREWORD ix

PREFACE xi

INTRODUCTION 1

The Need for New Methods, 2
The Product Development Cycle, 3

HUMAN FACTORS METHODS IN RESEARCH AND PRODUCT DESIGN 4

Analysis: Gathering Ideas, 4
Design: The Initial Design, 6
Formal Analysis of the Initial Design, 10
Building a Prototype, 11
Prototype Testing with Users, 12
Redesign, 16
Implementation: Monitoring Continued Performance, 16

OTHER METHODS 18

ADVANCES AND SUCCESSES 21

FUTURE METHODS 22

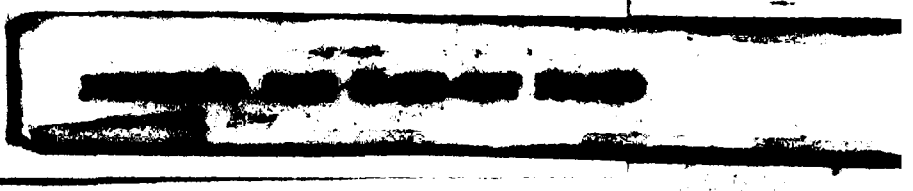
CONCLUSION 25

REFERENCES 26

FOREWORD

The Committee on Human Factors was established in October 1980 by the Commission on Behavioral and Social Sciences and Education of the National Research Council. It is sponsored by the Office of Naval Research, the Air Force Office of Scientific Research, the Army Research Institute for the Behavioral and Social Sciences, the National Aeronautics and Space Administration, and the National Science Foundation. The principal objectives of the committee are to provide new perspectives on theoretical and methodological issues, identify basic research needed to expand and strengthen the scientific basis of human factors, and to attract scientists both inside and outside the field to perform needed research. The goal of the committee is to provide the solid foundation of research as a base on which effective human factors practices can build.

Human factors issues arise in every domain in which humans interact with the products of a technological society. In order for the committee to perform its role effectively, it draws on experts from a wide range of scientific and engineering disciplines. The committee includes specialists in the fields of psychology, engineering, biomechanics, cognitive sciences, machine intelligence, computer sciences, sociology, and human factors engineering. Other disciplines participate in the working groups, workshops, and symposia organized by the committee. Each of these disciplines contributes to the basic data, theory, and methods required to improve the scientific basis of human factors.



PREFACE

This report covers Software Human Factors.

Computers are pervasive in civilian and military equipment systems. The compatibility of computer-based devices and human users is predominantly dependent on the characteristics of the software. The term *software human factors* refers to the process of designing software to be effective for human use, i.e., easy to learn and use, productive, and efficient. However, no specific efforts have been made to operationally define the objectives of software human factors--a necessary step both to focus research goals and to provide a framework for development of general application principles.

& xii

While a large amount of research has been performed on software features related to ease of use or user compatibility, most of these studies have been limited to a few features investigated in a specific context. Consequently, results from different studies cannot be integrated, and it is hard to draw conclusions that can be generalized to other situations. Overriding problems in the development of principles of software human factors are the lack of knowledge of how research on software human factors should be conducted and a paucity of techniques for measuring performance. For example, little is known about how to collect user data on "ease of learning," how to define errors, how to record complex response-time metrics, and how to measure user satisfaction.

Researchers interested in the development of principles for the design of user-compatible software have great need for guidance in both research methods and performance measurement techniques. As an initial effort to fulfill this need, the committee conducted a two-day workshop to bring together highly qualified researchers with knowledge

about how to design software to be usable based on studies in diverse fields.

The Workshop on Software Human Factors was convened in June 1983 in Washington, D.C. The impetus for the workshop grew directly from the review of the state of research and practice in human-computer interaction in the committee's 1983 report, Research Needs for Human Factors. The workshop had three goals:

- To identify current methods used to design and evaluate human factors aspects of software, including overall design and methods for collecting data on user performance;
- To ascertain what we know from software research results that we did not know 10 years ago; and
- To identify new research methods that are needed, both to develop design principles for software and to discover how users understand software systems.

A group of 14 nationally recognized, active researchers in the field of human-computer interaction from both industry and academia were invited to participate in the workshop. These workshop members represented a variety of pertinent disciplines, including human factors, cognitive psychology, computer science, experimental psychology, social psychology, and business administration. The relevant bodies of knowledge represented by the participants include experimental design and data analysis, human performance measurement, software design, information processing, learning, and attitude assessment. Prior to the workshop, participants prepared short, informal position papers on the issues for distribution. To accomplish the goal of collecting the desired knowledge about the design of software, the group spent two days listing both design and evaluation methods currently in use for the product development of good software and relevant research methods for understanding basic issues in user-software interaction; describing each method and constructing a list of references in which these methods are used; categorizing methods according to their uses in various stages of software product development or in more basic research; and suggesting new methods and techniques, designating their possible uses, and indicating which appear to have high near-term payoff.

The technical aspects of the workshop were organized by committee members Nancy S. Anderson and Alphonse Chapanis. The meeting was chaired by Nancy Anderson.

The report that follows, edited by Nancy Anderson and Judith Reitman Olson, is based on discussions from the workshop and written materials and references contributed by the participants during and subsequent to the workshop. Special appreciation is extended to Robert T. Hennessy and M. Jeanne Richards, formerly of the committee staff, for their contributions in making the sessions productive and pleasant; to Stanley Deutsch, study director of the committee, for his contributions to the organization and preparation of the report; to Christine McShane, of the Commission staff, for editorial support; and to Anne Sprague, administrative secretary, for secretarial and administrative support. They all helped to usher this report to publication.

Nancy S. Anderson, Chair
Workshop on Software Human Factors

INTRODUCTION

At present, software for specific applications and user-computer interfaces are aggressively developed in industry, but they are designed largely with only the designer's intuition as guide and often without empirical testing with end users. Two observations made in a popular software magazine point out the resulting problem:

> The computer systems and software we have today are too damn complicated for the end user. There is too much to learn, too many fiddly details, too much jargon, too much said that shouldn't be and not enough said that should be . . . (A. Johnson-Laird, Software News, April 1982).

> Data processing still has one ongoing problem to solve: the end user's dissatisfaction with today's systems. The entire industry has been grappling with this problem of ergonomics, or the interface between human and machine. In the case of data processing, ergonomics involves the development of "user-friendly" systems which can be operated by the user at the terminal and which generate results that the user can understand and utilize (M. Parks, Software News, February 1983).

Because of such difficulties, some industry and academic research groups are developing an interest in gathering and building appropriate guidelines from basic research and incorporating these guidelines and observations of users' behavior into the design process. A new field has emerged called software psychology or the psychology of human-computer interaction. It is in a very exciting state--a relatively new amalgam of

experimental/cognitive psychology, computer science, business, and engineering.

The field is growing in a variety of sectors. There are more human factors groups in industry than ever before. Approximately 50 universities in this country and abroad have PhD programs in human-computer interaction, which are housed in psychology, computer science, social sciences, engineering, business, and English departments (Mantei and Smelcer, 1984). Many more schools offer one or more courses in the area. The Association for Computing Machinery has a Special Interest Group for Computer-Human Interaction (SIGCHI). The Human Factors Society has a group called the Computer Systems Technical Group, which is concerned with human factors aspects of interactive computing systems, the data processing environment, and software development. Consumer demand for computers is increasing at a rapid pace, and many schools are acquiring computers for tutoring and the word-processing and mathematical tools that they provide. The systems that sell are those that provide the right usability and functionality--that provide the right design for the end user.

THE NEED FOR NEW METHODS

Designing systems to fit the end user is a difficult process. The field is searching for new methods. Classical experimental designs (e.g., controlled factorial designs) may not be appropriate for industrial settings in which cost-effectiveness and timeliness are major concerns. However, tests of single, intuition-driven designs with users, measuring their performance and satisfaction, do not advance our general knowledge about designs and do not indicate why certain features are good or bad.

There are, however, hybrid methods being used in industry, and new, more complex laboratory tests being constructed to assess users' performance in and understanding of complex systems. These methods are described below, along with their advantages and disadvantages and where they fit into the product development cycle. Each method is annotated with references to a few key articles that report its use.

THE PRODUCT DEVELOPMENT CYCLE

Software products are typically developed in three general stages:

1. Analysis--the product's functionality and initial hardware/software constraints are determined, analysis is made of the product's projected costs and benefits, and a development schedule is projected.
2. Design--the product is designed, first at the level of functional specifications and later in complete detail, then coded and tested, ending with a running system.
3. Implementation--the product is distributed and installed in its final locations, and users are trained and then operate the equipment.

At all three stages human factors considerations appear:

1. In assessing users' needs and capabilities during the analysis phase;
2. In designing and redesigning the system with human factors principles of usability, and in testing prototypes with end users during the design stage; and
3. In monitoring use of the system after its implementation, gathering information for redesign to correct errors or to add new, useful features.

In what follows the methods appropriate to each of these stages are described. These methods, or their variants, are useful for both laboratory research and industry. They may be used in the slower, more controlled environment of the laboratory, where research is designed to study people's performance on complex tasks. And they contribute equally to design and evaluation in industry, where timeliness is frequently considered to be more important than the ability to generalize from the results.

ANALYSIS: GATHERING IDEAS

The ideas behind products typically arise from three major sources: from the redesign of an existing product, from an identified need in the marketplace, and from a new technological capability that provides a useful new function to users. Information about the success of existing products can be obtained either by asking their users for their opinions and uses of the systems or by gathering unobtrusive data about their use. Information about a new product can come from reports of needs from potential users.

Reports from Users

Questionnaires and interviews are the most common methods for gathering information about the success of a product or the needs for new functions or a new product. Both questionnaires and interviews are good methods for eliciting information about how a person goes about his or her work, what aids or tools he or she uses or desires, what kind of knowledge or training is required to do the work, what difficulties he or she reports about the work, where the work originates and where it goes, what interactions are necessary with other people to do the work, and how the user thinks the work process could be improved. Questionnaires are more rigid in format than interviews, since interviews can go where the interviewee leads, often uncovering unanticipated new information. The principal disadvantage of interviews, however, is that they are time-consuming; only one person can be interrogated at a time. By aggregating information from

a number of interviewees or questionnaires, one can construct a general picture of users' needs and construct some tentative system concepts for helping the users do their work (Kelley and Chapanis, 1982; Rosson, 1983).

Diaries provide a similar form of informal data gathering and are used to uncover the needs and capabilities of the potential users of a new product. Data about work can be gathered in detail over a long period of time, especially about how much time particular kinds of activities take and their sequential dependencies. Because a shorter time elapses between the occurrence of an event and its report, diaries give a more accurate record of actual activity than retrospective reports in questionnaires and interviews (Mantei and Haskell, 1983).

A common marketing technique for gathering information about existing or potential users' needs is the *focus group*. Instead of interviewing a single user at a time, groups of users who are either similarly trained or who share common goals are first told about some potential capabilities of a system, then asked to discuss how they might find uses for these capabilities. Occasionally active brainstorming from these sessions generates very good ideas. The same kind of method is used to collect opinions about an existing product and to ask for suggestions for improvements. Often designers will gather expert users of a system and ask their opinion about how to improve the system or how to design a new, computer-based tool for aiding their work (Al-Awar et al., 1981). The advantage of such methods is that the participants stimulate each others' thoughts, uncovering ideas or suggestions they may not have thought of individually. That is also its disadvantage: a participant's true opinions can be swayed by group pressure.

Inferring Needs from Natural Observation

One of the main drawbacks of the methods listed above is that they rely on users' perceptions of their needs and capabilities. Sometimes new products meet needs unforeseen by their users; sometimes users, either consciously or unconsciously, distort their daily work activities and feelings about existing working conditions. In such cases, it may be better to collect information, not by asking users, but by watching their behavior and inferring their needs and capabilities from their activities.

Two methods are often used to collect information about users' behavior in natural work settings. In the case of *activity analysis*, an observer watches and records certain behaviors of the workers. The data may be collected by direct observation or by analyzing video or film recordings. Individual samples of categorized activities are aggregated into activity frequency tables, graphs, or state transition diagrams. Such performance analyses are particularly useful in assessing the changes made in work by comparing activity before and after a new system or design change is implemented (Hartley et al., 1977; Hoecker and Pew, 1980).

Logging and metering techniques involve observations of what a user does with a system, but the measurement is embedded directly into the software. These procedures can include a simple record with a time-stamp of every interaction that a user makes with the computer, or it can involve a complete hard copy representation of a sequence of particular display frames. Powerful logging and metering software can also categorize certain recognizable events and summarize their times. For example, one could summarize such events as time to complete a task, user and/or system response time, and frequencies and types of errors.

Logging and metering procedures are typically embedded in the operational software. Where there are limits to the access to such software, one can connect a second computer in tandem to the first and direct data about the user's activities to it, in essence providing a "passive tap." In this way, logging does not interfere with system response times, and information about the user inputs and the system responses can be recorded in detail for future use (see Whiteside et al., 1982; Goodwin, 1982).

DESIGN: THE INITIAL DESIGN

Designers go through two stages in constructing an initial design, either implicitly, driven by intuition or experience, or explicitly, using some or all of the detailed tools described below. First, the designers decide what the user is going to do, conducting an informal or formal task analysis. Second, they specify what the interface will look like and what the dialog will consist of. There are a variety of methods that apply to this stage, where designers use informal or

formal guidelines, consult end users, or have some theory-based judgments to draw on.

Determining What the User Needs to Do

The most common form of analyzing the user's activities is called a *task analysis*. Task analysis is the process of analyzing the functional requirements of a system to ascertain and describe the tasks that people perform. It focuses both on how the system fits within the global task the user is trying to perform (e.g., prepare a report of a projected budget) and what the user has to do to use the system (e.g., access the application program, access the data files, etc.).

Task analysis has two major aspects: the first specifies and describes the tasks, and the second, and more important, analyzes the specified tasks to determine such system or environmental characteristics as the number of people needed, the skills and knowledge they should have, and the training necessary. The first step involves decomposition of tasks into their constitutent subtasks and annotating each subtask for its essential elements and their interdependencies. The second step involves examination of the actual tasks and interdependencies, assessing how difficult each is, what knowledge is required, where the information resides, etc. Results of task analyses are used not only in writing functional specifications for a particular application, but also for assigning work to groups of workers, arranging equipment in an efficient configuration, determining task demands on people, and developing operating procedures and training manuals (see Bullen and Bennett, 1983; Bullen et al., 1982).

Specifying the Initial Design

An initial system or interface design is constructed next. With the global tasks the user has to perform specified as above, the designer groups the subtasks according to logical function from the perspective of the user but tempered by system/hardware constraints. Then the actual interface or system details come from three sources: design guidelines or principles, intuitions of the designer sometimes aided by intuitions of the users themselves, and theory-based judgments.

In generating an initial design, the designer can address existing *design guidelines* for general prescriptions of how to specify particular components of the interface. For example, if the interface has a menu, the guideline may prescribe that the alternatives should be listed by order of frequency of use or cluster them according to functional similarity, rather than displayed alphabetically or randomly. Current design guidelines (e.g., Woodson and Conover, 1966; Van Cott and Kinkade, 1972) include prescriptions about such topics as the readability of type fonts, the brightness levels of display screens, keyboards designed to fit hand shape and function, and rules for making abbreviations and symbols (see also Schneiderman, 1982; Smith, 1982).

Current guidelines, however, are more concerned with perceptual and performance characteristics than with the cognitive properties of the interaction. Thus, they would prescribe appropriate type fonts, but not what words these fonts should express to the user to suggest the appropriate analogy for performing the task on the system. There are several major caveats in the use of design guidelines: the prescriptions or recommendations contained may have been derived from situations or research not applicable to the system being designed; new or unaccounted for variables may interact in unanticipated ways; and current guidelines do not always publish the source of the recommendation, whether it was generated by a controlled laboratory study or derived from the collected wisdom of experience. Guidelines have to be applied with care.

Though design guidelines have their flaws, they are very useful in placing a particular new design in a setting of conventional wisdom. Often the designer, skilled in interacting with systems and cognizant of the end tasks that are being supported in this design, cannot foresee the difficulties the new user will have with the system. Design guidelines provide suggestions to the designer that will in many cases be better than those based solely on intuition. (For a recent version of guidelines, see Smith, 1984.)

The *skills and knowledge of users* themselves can be used to advantage by incorporating users in the design team. Users can provide some critical insights about how they think of the task and thus the system (e.g., what kinds of information should be accessible when, what the screens should look like to mimic the original, a noncomputer version of the task, what commands ought to

be called). They know the procedures and terminology and, with proper support, can contribute to the design and layout of forms and menus as well as act as critics of the design. Gould and Lewis (1985) and Miller and Pew (1981) provide examples of the involvement of users in the design process. Other ways in which the sophisticated user can be involved in the design of software systems can be found below in the section on prototype testing with users.

A third source of information about the original design specification is psychological theories. *Theory-based judgments* can constrain aspects of a design or suggest promising areas of investigation. For example, theories of color contrast can provide insight into the appropriateness of certain combinations used in screen highlighting or predict the readability of a new monochrome display color. Because Fitt's Law accounted for movement time for placing a cursor in a desired position with a mouse and for placing the appropriate finger on a desired key location, two conclusions follow: the invention of faster pointing devices was unlikely to increase performance and the design of keyboards with larger peripheral key caps would increase the accuracy of keying (Card et al., 1978; Card et al., 1980b).

Part of the difficulty in constructing a design and analyzing its usability has to do with how the interface is specified. Verbal descriptions of how a system works are particularly unsuited for conveying the flow of an interaction and the choices the user has at each point. Several specification languages or formats have been explored recently not only to serve as a way of conveying to those who actually build or code the system what it will do but also as a way of concretely specifying the system to analyze its usability.

One way to specify the interaction is to use an interactive tool kit called a *human-computer dialog management system*. This system guides the definition of the interaction language that describes the actions of the user and the system and the screen formats displayed at each moment. Hartson et al. (1984), Jacob (1983), and Wasserman (1982) provide good examples of this kind of interface definition.* A second format for displaying

*This is also a system that allows rapid embodiment of the functioning of a new, developing system and thus is a tool for rapid prototyping.

what the system does at each state is a *state transition diagram*, recently used as a description of a system's workings in Kieras and Polson (1983).

DESIGN: FORMAL ANALYSIS OF THE INITIAL DESIGN

Once an initial design is specified, even if it is a partial design, it can be subjected to several kinds of scrutiny. The goal in this analysis stage is to make the initial design as good as possible before it is made into the prototype for user testing. Three methods aid in this process: structured walk-throughs, decomposition, and task-theoretic analytic models.

Structured walk-throughs involve construction of tasks that a user carries out on a simulated system. The user tries out the system by going through the task, step by step, screen by screen, command by command. This can be done with the design as specified in a number of different formats, using an experimental simulation of a prototype or even with the experimenter presenting paper and pencil figures of the screens, menus, and commands in the appropriate sequence. The technique helps to identify confusing, unclear, or incomplete instructions, illogical or inefficient operations, unnatural or difficult procedures, and procedural steps that may have been overlooked because they were implicitly rather than explicitly defined. Gould et al. (1983), Ramsey (1974), Ramsey et al. (1979), and Weinberg and Friedman (1984) provide examples of the use of structured walk-throughs.

A second kind of formal analysis, called *decomposition*, is proposed in Reitman et al. (1985). In this analysis, the major components of the design are separated and analyzed for their impact on cognition. The picture displayed on the screen, for example, is assessed for how it helps or hinders the user's ability to perceive meaningful relationships or the system model. The commands are assessed for their load on long-term memory, how easy they are to remember, and how confusable they are among each other. For each component, a second design alternative is constructed to fit within the general guidelines of usability. Then, through discussion and debate, the design team decides which alternative of each component is the better design. This method encourages careful scrutiny of the proposed design and often encourages designers to specify better interfaces before the first prototype is built.

The third kind of formal techniques invoke *task-theoretic analytic models*. These models provide representations and analyses that assess, for example, which parts of a metaphor aid performance and which do not (Douglas and Moran, 1983) and how big the user's short-term memory load is at each step of the interaction (Kieras and Polson, 1985). Prime examples of these techniques include metaphor analysis (Carroll and Thomas, 1982; Carroll and Mack, 1982), assessment of mental models (deKleer and Brown, 1983; deKleer and Brown, in press; and others in Gentner and Stevens, 1983), development of production rule systems that represent the user's knowledge of the task (Kieras and Polson, 1985), object/action analysis (called "external/internal task mapping" by Moran, 1983), the GOMS model (Card et al., 1980b; 1983), and formal grammar notation systems (Reisner, 1981a, 1984; Blesser and Foley, 1982).

These task analytic models are very useful tools. However, none of them yet encompasses all of the cognitive aspects of the interaction; each focuses on one or more important aspects. These methods require training to use and often take a long time. However, they all have the advantage of being based on sound theories of human behavior and can provide important analysis of usability before any coding of software or running of subjects is contemplated. There is a trade-off, then, between time spent in analysis and time spent testing users in the laboratory or the field. The hope embodied in this approach is that as the science of user-interface design grows, analytic tools will improve to the point of making the actual user testing of designed systems merely a last, short check of a good, finished design.

DESIGN: BUILDING A PROTOTYPE

Three methods provide simulations or quick versions of significant aspects of a new system so it can be tried by actual users. The methods are called facading, the Wizard of Oz technique, and rapid prototyping.

Facading is the technique of quickly and inexpensively building a simulation of the external appearance (i.e., the "facade") of a system's interface. Its advantages are that it is quick and relatively easy; the target system's underlying complexity and/or final computational capability is "finessed." To be maximally beneficial, the facade must embody some level of the functional

capability of the final target system. It does not just generate a series of static snapshots of the system but rather includes the control structure, flow, or connectivity of the final system. Hanau and Lenorovitz (1980) and Lenorovitz and Ramsey (1977) provide good examples of the use of this technique.

A variant of the facading technique is the *Wizard of Oz* technique. Instead of having the computer embody the simulated system, hidden human operators intercept user commands and provide output back to the user. Often the technique is used to test a new interface language: the hidden human operator intercepts the new commands, translates them into the real system commands, and, after receiving output from the real computer system, retranslates them back to the tested end-user (see Gould et al., 1983; Gould and Boies, 1978; Ford, 1981; Kelley, 1983; Wixon et al., 1983).

Rapid or fast prototyping are terms applied to the more formalized building of a prototype in a hurry. The speed of building a running system depends mainly on the underlying supporting software, which makes the specific prototype programmable from existing modules. Ideally, the prototype programming language separates elements of the dialog from the actual implementation software. For example, the designer can specify the placement of the command input line or the menu choices variously without having to program new modules to execute these different input formats. One of these, the "dialog management system," is under development by Hartson and his colleagues (Hartson et al., 1984; Yunten and Hartson, 1984); another system is described in Wasserman (1982) and Wasserman and Shewmake (1982). Another project that uses rapid prototyping methods is reported in Hayes et al. (1981).

DESIGN: PROTOTYPE TESTING WITH USERS

When a prototype of some form has been built, actual users are then brought in to use the system and report their opinions about it. These tests can vary greatly in how well controlled their designs are and how representative the set of tested users are of the final population of users. Moreover, users are asked to perform several kinds of tasks, some testing the normal, frequent tasks that regular users will be expected to perform, others testing those subtasks thought to be especially difficult

either for the system (e.g., those producing long system response times) or for the user (e.g., the longest sequence of commands for a particular type of task). Prototype tests differ in what kinds of data are taken from the user--times and errors, thinking aloud protocols, or attitudes.

Experimental Designs

Field tests to evaluate systems are fashioned after laboratory tests common in the academic field of experimental psychology. In general, they require the comparison of at least two systems, systems that differ in only one component or variable. Measures are designed to reflect the performance attributable to the effects of that variable, and subjects are chosen to be representative of the population of end users. Of particular importance are various techniques for controlling irrelevant variables. For example, one must ensure that measures of intelligence of the test subjects do not differ across both conditions, affecting the results in addition to the effects of the independent variables.

Often the rules of good experimental design are violated in the interest of proceeding quickly. Subjects who are different from the end users but more available may be tested; comparisons may be made between two systems that differ on more than one variable; measures may be taken that are less sensitive than those that will directly test why performance on one system is better or worse than another; occasionally only one system is tested and performance on it is measured against some predetermined standard (e.g., a 10-minute rule for time to learn a system). The closer the test is to good experimental design, the more quickly the findings can advance knowledge about the important aspects of good human-computer interface. However, as is often the case in development, the goal is not ultimate knowledge but rather global assessment of the adequacy of a particular interface or system. A compromise design procedure is described in Reitman et al. (1984). The use of experimental design is found in Ledgard et al. (1981), Reisner et al. (1975), Reisner (1977, 1981b), and Williges and Williges (1982).

One variant from controlled experimental evaluation that has been found useful in the development of interfaces is called *quasi-experimental design*. These

designs involve capturing data at several time intervals, typically of durations measured in weeks or months. Sometime during the data capturing intervals, a change or a modification of a system is introduced; the data being captured are expected to reflect the impact of this change. Some of these quasi-experimental designs allow for comparisons with a control group. These designs are hard to control, since the investigator must typically take existing groups of users, giving one the change and the other no change. Inherent differences in existing groups is a major worry in evaluating the results. A complete description of this technique can be found in Cook and Campbell (1979); Koltum (1982) and Rice (1982) provide good examples of this method.

Selection of Tasks to Perform

There are two reasons one has users try out a prototype system: to identify points of difficulty for the user so that those points can be redesigned and to measure standard use of the system, so that later changes in hardware can be assessed or so those concerned with the staffing of a large operation of users can determine how many people will be needed. For the first purpose, tasks are selected that stress the system and the user, generally called critical incidents. For the second purpose, tasks are selected to estimate basic characteristics of the system's use, called benchmark tests.

In terms of *critical incidents*, the goal is to set up situations or tasks that have been shown historically to tax the user and/or the system and are sufficiently important that they can make the difference between success or failure on task or system performance. One might, for example, require the user to access items distant from what is being presented on the current screen or to perform a long command sequence, to determine the loads of this part of the design on the user's ability to imagine the stored information's underlying structure or the mnemonic characteristics and grammatical rules implied by the command sequences. The goal is to set up situations in which the data will tell the designers something about the limits of human or system performance. These tasks are illustrated in the work of Al-Awar et al. (1981), Kelley and Chapanis (1982), and Flanagan (1954).

In *benchmark* tests, the goals are quite different. The designer wants to measure the likely performance times and errors expected in normal use. The tasks are not designed to tax the system or the user, but rather to be representative of the kinds of frequent tasks the system will normally support. Typically, tasks are constructed to measure the expected amount of time it takes a new user to learn a system, the amount of time it takes the user to perform a set of predefined tasks, and the amount of time it takes the system to respond to a user's request. A good study that illustrates the use of this method is that of the evaluation of eight text editors by Roberts and Moran (1983). A study of data-base interfaces using benchmarks was done by Mantei and Cattell (1982).

Kinds of Data Collected

There are four major kinds of data collected in tests of systems: the time it takes to perform a task, the frequency and kinds of errors, the goals and intentions of the users, and the attitude of the user.

The amount of time a task takes (either how long an entire task takes or how long *each* successive keystroke takes) reflects the time it takes the user to perceive inputs, categorize and plan appropriate actions, and execute proper responses. Error frequencies and types reflect the difficulties users have with these processes and often point to the cause of the error (whether the error response is similar to one in a similar plan, was generated from confusion with a similar screen, has a label that sounds the same as another, etc.) A simple analysis of users' *times and errors* is found in Reisner et al. (1975) and Reisner (1977). A comprehensive analysis of users' times is found in Card et al. (1980b, 1983). Other uses of times and errors can be found in Boies (1974), Rosson (1984), Sheppard and Kruesi (1981), and Thomas and Gould (1975).

A more thorough, complicated kind of data to collect during evaluation involves the user's *thinking aloud* while performing the task. Typically the user is video- and sound-recorded while he or she is performing the tasks. The recording captures what is said and done, what is displayed on the screen, what sections of the documentation are being examined, what parts of the task instructions the user is reviewing, etc. The most

complete protocols ask the subjects to verbalize their intentions, what their goals are, and what current plans they have about reaching their goals. Other behavior is directly observable; thoughts and plans typically are not. This method has been used by Mack et al. (1983), Carroll and Mack (1982), and Card et al. (1980a) in their studies of skilled text editing. More complete descriptions of the technique and its advantages and disadvantages can be found in Lewis (1982), Olson et al. (1984), and Ericsson and Simon (1980).

A third kind of data collected in evaluation sessions is the *users' opinions* about the system's ease of use and functionality. A common instrument used to scale users' global attitudes about the system is the evaluation component of Osgood et al.'s (1957) Semantic Differential (see Good, 1982, for an example of its use). Questionnaires and interviews also tap users' reactions to particular components of the system. One problem with users' reports, however, is that they are typically distorted by their experience with other, similar systems. Or a user may have difficulty separating components of the system such; for example, a user who has a very difficult time using a system may report that he or she likes it a great deal, recognizing how much easier it is to perform the task on a computer compared with previous manual methods.

Redesign

Typically as the prototype of the original design is tested, errors are found and revisions suggested. The methods appropriate to the initial design are appropriate also at the stage of redesign. This part of the design process iterates through "fixing" and "testing" until either an acceptable level of performance is reached or the deadline for developing the system is reached.

IMPLEMENTATION: MONITORING CONTINUED PERFORMANCE

Just as data were collected in the original conception and analysis phase of product development, data are collected on the system as implemented. At this stage, activity analyses, diaries, logging and metering, and questionnaires and interviews are all appropriate methods for assessing whether the product as designed is perform-

ing as predicted in the final environment. If problems are found in the field, either small corrections are made in the code (e.g., changing what a command is called is easy to change in the code but can have an enormous impact on the ease of use), or a redesign is called for, sending the product design process back to prototype development or fully back to the top of the cycle.

OTHER METHODS

Three additional methods are worth mentioning, though they do not fit neatly into the scheme above. They include the dialog specification procedure, experimental programming, and case studies.

The *dialog specification method* is a global procedure that cuts across the first several steps outlined above. It is a procedure that prescribes a method for developing an interactive dialog with a system and sets a design standard. The method includes task analysis and flow charting of user activities as well as standard means of communicating the specific design requirements to the programmer. The design standard describes acceptable screen layouts, interactive devices and how they are to be used, acceptable command language syntax, etc., down to a level of detail compatible with the specificity of the range of applications to which it is intended to apply. For example, if all designs concerned telephone management applications, the specification would deal only with the range of tasks in this domain. These specifications are built from human factors principles as well as accumulated data from user testing. Pew et al. (1979) describe this method more fully.

Experimental programming is similarly a more global method for designing systems and interfaces. It is a more flowing, adaptive technique involving users, designers, and programmers (sometimes all in the same person). Someone builds a prototype of a new system with some fraction of the functionality and some fraction of the user interface in place. This prototype is then used by a variety of programmer/users who generate suggestions for new features and suggestions for revisions for existing functions. As many suggestions as possible are incorporated into the prototype; the good features

survive, poor features disappear. Occasionally, when new features are incompatible with the old, a competing prototype is built. Sometimes someone merges the most popular ideas from both. This method is very informal. The only rules for its application are that everyone's opinion get a fair hearing and that anyone in the community can implement a change.

This method allows for progressively better understanding of the application as well as the computation and interface requirements. Its weakness lies in its casual nature and that it relies on the opinion of users, most of whom are programmers; its strength lies in its exploratory, evolutionary, democratic nature. One well-known product that benefited from experimental programming is the EMACS text editor (Stallman, 1980), which pioneered such concepts as user-customization, on-line documentation, and a particular command style. In addition, Teitelman (1972) used experimental programming to develop the concept called DWIM ("Do What I Mean"), which included a set of facilities that automatically corrected predictable errors.

A third global technique goes under the rubric of *case studies*. Case studies involve observation and analysis of a singe user, group, or project. The information collected may range from informal, subjective impressions to detailed quantitative data. Because case studies involve no comparison or control group, they are not very useful in inferring causality. As a result they are not appropriate for building a data base of basic research results from which to construct theories and principles. They can, however, be extremely useful for gaining insights when one is first investigating an area of interest and for providing concrete demonstrations of the use of new methods and tools.

An example of a case study in which new insights were gained about a domain involved the use of the Ada system. The purpose of the study was to understand the problems that are likely to arise when the system is first introduced into an organization (Bailey et al., 1982). A second case study involved a demonstration of new methods for designing systems to be embedded in special purpose hardware, such as airplanes and tanks (Britton et al., 1981). The documentation and related products produced by this case study provide examples that others may use in trying to apply the methods to their own software projects. Brooks (1975) documents the use of a case study in a large computer programming project. And, the

case study by Baker (1972) was extremely influential in leading the structured programming revolution. Others include Gould and Boies (1978, 1983, 1984), and Heninger (1980).

ADVANCES AND SUCCESSES

Over the last 10 years, it has become clear that research on the issues surrounding human-computer interaction is worth doing. The design of the human-computer interface makes a marked difference in users' performance. Software products exist that embody well-designed interfaces derived from human factors input: the Xerox STAR, Apple LISA, and MACINTOSH work stations and the Rolm and IBM mail systems are examples. In addition, major changes in the design of the telephone directory assistance system, as well as original designs of telecommunication control devices, were a result of human factors studies.

Human factors research has also shown the usefulness of some important generic display and control devices: the partitioning of screens into windows, icons for the control of operations and the display of objects, better help messages, and better defined response and function keys. In addition, more is known about users' limitations and adaptability.

Human factors design is also influencing documentation and training for software use (Felker, 1980). Because software is more available to a variety of users, there is an increased awareness by the public of the need to make software easy to learn and use.

FUTURE METHODS

Although we have catalogued a variety of methods to be used in the software design and research process, some needs for information are still unmet. The research needs fall roughly into three categories of needs: new theories, new representations, and new data collection and analysis methods.

THEORIES

Three particular kinds of theories are seen as needed. *Automation theories* would tell us what should be automated and what should be assigned to the human processor. Such theories would also prescribe an appropriate mix of automation and human control. Some seeds of theories are suggested in the field of supervisory control and in office analysis techniques, but a more explicit theory is needed to prescribe the best mix of human and computer processing.

Theories of individual differences would tell us about the different kinds of computer support required and desired by different user populations. Special continuing interest focuses on the differences between naive or casual users and expert or dedicated users.

Theories of standardization would tell us about which aspects of a system should be standardized for all users (as in the basic control devices in an automobile) and which can be customized for adaptation by and for specific users.

In addition, two taxonomies are needed: a characterization of the kinds of tasks for which software can be built (so that design prescriptions can be tied, perhaps, to particular classes of tasks) and a characterization of

the kinds of users that use software applications (related to the theories of individual differences described above). The partial taxonomy of human-computer interface tasks advanced by Lenorovitz et al. (1984) provides a baseline for this effort.

REPRESENTATION

Many of our analyses outside the testing of a working system with real end users require some specification of what the system can do, what the user knows about how the system works, and how the user conceives of the task. There is thus a need for better representational schemes than those now being used. One such scheme would describe a complex system so that documentation and training could be better designed. Another would represent exactly how a system works--the interface, dialog, communication, or transaction--so that the design could be both analyzed for its fit to users' needs and capabilities and conveyed to those who have to program it.

We need techniques for inferring what a user currently understands of a system, a method for extracting the appropriate information from the user and for displaying the resulting understanding or "mental model." These techniques are as useful in basic research on the performance of complex tasks as they are in the applied design process. (A report of the Committee on Human Factors' workshop on mental models in the use of information systems is scheduled for publication in 1985.)

DATA COLLECTION, MEASURES, AND ANALYSES

Although we have a rich variety of measures to collect from users interacting with a system, we have no direct measures of the user's affect nor do we collect any of the neurophysiological responses that accompany intense work, frustration, and satisfaction. In addition, there is a need for better hardware tools for collecting logging and metering information without slowing the system that the user normally interacts with. More specific methods are needed for analyzing the mountain of data that comes from protocol analysis, not only in deducing how the user is satisfying his or her task goals and subgoals, but also in deducing ongoing memory and perceptual loads on the user and how the user compensates for them in per-

forming the task. Our task analysis methods need to be expanded to include more cognitive aspects of the user's performance, his or her memory, language, and perceptual aspects.

Research methods considered most likely to produce high payoff in the near future include:

- Representations of the users' understanding of a system;
- Representations of a dialog to convey the design to programmers;
- More comprehensive task analyses that include memory, perceptual, and language considerations as well as timing and error predictions; and
- Hardware advances that allow the collection of logging and metering data for tapping the current use of a system.

CONCLUSION

The field of software human factors is rising in its research needs faster than the scientific data base is growing. Additional basic research is clearly needed. Educational programs are now training future researchers and practitioners in this field. Data in laboratories and industry need to be collected more systematically and disseminated more widely. As a compendium of current methods, their descriptions and evaluations, and references to existing literature that use these methods, this report should then help coalesce the field and move it toward fruitful work in the future.

REFERENCES

Al-Awar, J., Chapanis, A., and Ford, W.R.
1981 Tutorials for the first time computer user. IEEE Transactions in Professional Communication. PC-24(1):30-37.

Bailey, J., Basili, V., Gannon, J., Katz, E., Kruesi, E., Sheppard, S., and Zelkowitz, M.
1982 Monitoring an Ada Software Development Project. Ada Letters 2(July-August):58-61.

Baker, F.T.
1972 Chief programmers team management of production programming. IBM Systems Journal 11:56-73.

Blesser, T., and Foley, J.D.
1982 Towards specifying and evaluating the human factors of user-computer interface. Pp. 309-314 in Proceedings of the Human Factors of Computing Systems. New York: Association of Computing Machinery.

Boies, S.J.
1974 User behavior on an interactive computer system. IBM Systems Journal 13:2-18.

Britton, K.H., Parker, R.A., and Parnas, D.L.
1981 A procedure for designing abstract interfaces for device interface modules. Proceedings of the 5th International Conference on Software Engineering. Orlando, Fla: IEEE.

Brooks, F.P.
1975 The Mythical Man Month: Essays on Software Engineering. Reading, Mass.: Addison-Wesley.

Bullen, C.V., and Bennett, J.L.
1983 Office Workstation Use by Administrative Managers and Professionals. IBM Research Report RJ 3890.

Bullen, C.V., Bennett, J.L., and Carlson, E.D.
1982 A case study in office workstation use. IBM Systems Journal 21(3):351-369.

Card, S.K., English, W.K., and Burr, B.J.
1978 Evaluation of mouse, rate-controlled isometric joystick, stop keys, and text keys for text selection on a CRT. Ergonomics 21:601-631.

Card, S., Moran, T., and Newell, A.
1980a Computer text-editing: an information processing analysis of a routine cognitive skill. Cognitive Psychology 12:32-74.
1980b The keystroke level model for user performance with interactive systems. Communications of the ACM 23:396-410.
1983 The Psychology of Human Computer Interaction. Hillsdale, N.J.: Lawrence Erlbaum.

Carroll, J.M., and Mack, R.L.
1982 Metaphor, computing systems and active learning. IBM Research Report RC 9636.

Carroll, J.M., and Thomas, J.C.
1982 Metaphor and the cognitive representation of computing systems. IEEE Transactions on Systems, Man, and Cybernetics 12:107-116.

Cook, T.D., and Campbell, D.T.
1979 Quasi-Experimentation: Design and Analysis Issues for Field Settings. Chicago: Rand McNally.

deKleer, J., and Brown, J.S.
1983 Assumptions and ambiguities in mechanistic mental models. In D. Gentner and A.S. Stevens, eds., Mental Models. Hillsdale, N.J.: Lawrence Erlbaum.
in press A qualitative physics based on confluences. In B. Moore and J. Hobbs, eds., Formal Models of the Common-Sense World. Norwood, N.H.: Ablex.

Douglas, S.A., and Moran, T.P.
1983 Learning text editing semantics by analogy. CHI-83. Pp. 207-211 in Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery.

Ericsson, K.A., and Simon, H.A.
1980 Verbal reports as data. Psychological Review 3:215-251.

Felker, D.C., ed.
1980 Document Design: A Review of the Relevant Research. American Institute for Research. Technical Report 75002-4/80, Washington, D.C.

Flanagan, John C.
1954 Critical incident technique. Psychological Bulletin 51:327-358.

Ford, William R.
1981 Natural Language Processing by Computer--A New Approach. Ph.D. dissertation. Department of Psychology, Johns Hopkins University.

Gentner, D., and Stevens, A.L. eds.
1983 Mental Models. Hillsdale, N.J.: Lawrence Erlbaum.

Good, M.
1982 An ease of use evaluation of an integrated document processing system. CHI 82. Pp. 142-147 in Proceedings of Human Factors in Computing Systems. New York: Association of Computing Machinery.

Goodwin, N.C.
1982 Effect of interface design on usability of message handling systems. Pp. 69-73 in Proceedings of the Human Factors Society. 26th annual meeting, Seattle, Wash.

Gould, J.D., and Boies, S.J.
1978 Writing, dictating, and speaking letters. Science 201:1145-1147.
1983 Human factors challenges in creating a principal support system--the speech filing approach. ACM Transactions on Office Information Systems 1(4):273-298.
1984 Speech filing--an office system for principals. IBM Systems Journal 23(1):65-81.

Gould, J.D., and Lewis, C.
1985 Designing for usability of key principles and what designers think. Communications of the ACM 28:300-311. New York: Association of Computing Machinery.

Gould, J.D., Conti, John, and Hovanyecz, Todd
1983 Composing letters with a simulated listening typewriter. Communications of the ACM 26:295-308.

Hanau, P.R., and Lenorovitz, D.R.
1980 A prototyping and simulation approach to interactive computer system design. Pp. 23-25 in Proceedings of the 17th Design Automation Conference, Minneapolis, Minn.

Hartley, C., Brecht, M., Pagersy, P., Weeks, G., Chapanis, A., and Hoecker, D.
1977 Subjective estimates of work tasks by office workers. Journal of Occupational Psychology 50:23-36.

Hartson, H.R., Johnson, D.H., and Ehrich, R.W.
1984 A human-computer dialogue management system. Pp. 57-61 in Proceedings of INTERACT '84, London. Amsterdam: Elsevier Science Publications.

Hayes, P., Ball, E., and Reddy, R.
1981 Breaking the man-machine communication barrier. Computer 14(3):19-30.

Heninger, K. L.
1980 Specifying software requirements for complex systems: new techniques and their application. IEEE Transactions on Software Engineering. SE-6(1):2-13.

Hoecker, D.G., and Pew, R.W.
1980 User Input to the Design and Evaluation of Computer-Assisted Service Delivery. Report #4358. Cambridge, Mass.: Bolt Beranek and Newman Inc.

Jacob, R.J.K.
1983 Using formal specifications in the design of the human-computer interface. Communications of the Association of Computing Machinery 26(4):259-270.

Johnson-Laird, A.
1982 Most software more complicated than needed. Software News 2(4):47.

Kelley, J.F.
1983 Natural Language and Computers: Six Empirical Steps for Writing an Easy-to-Use-Computer Application. Ph.D. dissertation, Department of Psychology, Johns Hopkins University.

Kelley, J.F., and Chapanis, A.
1982 How professional persons keep their calendars: implications for computerization. Journal of Occupational Psychology 55:241-256.

Kieras, D.E., and Polson, P.A.
1983 A generalized transition network representation for interactive systems. CHI-83. Pp. 103-106 in Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery.

1985 An approach to formal analysis of user complexity. International Journal of Man-Machine Interaction. In press.

Koltum, P.L.
1982 Evaluation of a Teaching Approach for Introductory Computer Programming. Ph.D. dissertation, Department of Computer Sciences, University of North Carolina.

Ledgard, H., Singer, A., and Whiteside, J.A.
1981 Directions in Human Factors for Interactive Systems. New York: Springer-Verlag.

Lenorovitz, D.R., and Ramsey, H.R.
1977 A dialogue simulation tool for use in design of interactive computer systems. Pp. 95-99 in Proceedings of the Human Factors Society Annual Meeting. Santa Monica, Calif: Human Factors Society.

Lenorovitz, D.R., Phillips, M.D., Ardrey, R.S., and Kloster, G.V.
1984 A taxonomic approach to characterizing human-computer interfaces. In Human-Computer Interaction, G. Salvendy, ed., Proceedings of the First USA-Japan Conference on Human-Computer Interaction. Amsterdam: Elsevier Science Publications.

Lewis, C.
1982 Using the "thinking aloud" method in cognitive interface design. IBM Research Report RC #9265.

Mack, R.L., Lewis, C. and Carroll, J.M.
1983 Learning to use word processors: problems and prospects. ACM Transactions on Office Information Systems 1:254-271.

Mantei, M., and Cattell, R.G.G.
1982 A study of entity-based data base interfaces. ACM SIGCHI Bulletin 14(1).

Mantei, M., and Haskell, N.
1983 Autobiography of a first-time discretionary microcomputer user. CHI 83. Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery.

Mantei, M., and Smelcer, J.B.
1984 Listing of doctoral programs in human-computer interaction. ACM SIGCHI Bulletin 16(2):12-40.

Miller, D.C., and Pew, R.W.
1981 Exploiting user involvement in interactive system development. Pp. 401-405 in Proceedings of the Human Factors Society Annual Meeting. Santa Monica, Calif: Human Factors Society.

Moran, T.P.
1983 Getting into a system: external-internal task mapping analysis. CHI-83. Pp. 45-49 in Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery.

Olson, G.M., Duffy, S.A., and Mack, R.L.
1984 Thinking-out-loud as a method for studying real-time comprehension processes. Pp. 253-286 in D.E. Kieras and M.A. Just, eds., New Methods in Reading Comprehension. Hillsdale, N.J.: Lawrence Erlbaum.

Osgood, C.E., Suci, G.J., and Tannenbaum, P.H.
1957 The Measurement of Meaning. Champaign-Urbana: University of Illinois Press.

Parks, M.
1983 Productivity tools enable users to obtain better (not more) code. Software News 3(2):22-23.

Pew, R.W., Rollins, A.M., and Williams, G.A.
1979 Generic Man-Computer Dialogue Specification: An Alternative to Dialogue Specialists. Bolt Beranek and Newman Inc., Cambridge, Mass.

Polson, P., and Kieras, D.E.
1985 A quantitative model of learning and performance of text editing knowledge. CHI-85. In Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery (in press.)

Ramsey, H.R.
1974 Plans: human factors in the design of a computer programming language. Pp. 39-41 in Proceedings of the Human Factors Society Annual Meeting. Santa Monica, Calif.: Human Factors Society.

Ramsey, H.R., Atwood, M.E., and Willoughby, J.K.
1979 Paper simulation techniques in user requirements analysis for interactive computer systems. Pp. 64-68 in Proceedings of the Human Factors Society Annual Meeting. Santa Monica, Calif.: Human Factors Society.

Reisner, P.
1977 Use of psychological experimentation as an aid to development of a query language. IEEE Transactions on Software Engineering SE-3(3):218-229.
1981a Formal grammar and human factors design of an interactive graphics system. IEEE Transactions on Software Engineering SE-7(2):229-240.
1981b Human factors of data-base query languages: a survey and assessment. Computing Surveys 13:13-31.
1984 Formal grammar as a tool for analyzing ease of use: some fundamental concepts. In J. Thomas and M. Schneider, eds., Human Factors in Computer Systems. Norwood, N.H.: Ablex.

Reisner, P., Boyce, R.F., and Chamberlain, D.D.
1975 Human factors evaluation of two data base query languages: SQUARE and SEQUEL. Pp. 447-452 in Proceedings of the National Computer Conference. Arlington, Va.: American Federation of Information Processing Societies Press.

Reitman, J.S., Whitten, W.B.,II, and Gruenenfelder, T.M.
1985 A general user interface for entering and changing tree structures, nested menus, and decision trees. In Proceedings of NYU Symposium on User Interface Design. Norwood, N.H.: Ablex.

Rice, Ronald E.
1982 Human Communication Networking in a Teleconferencing Environment. Ph.D. dissertation, Department of Computer Sciences, Stanford University.

Roberts, Teresa L., and Moran, Thomas P.
1983 The evaluation of text editors: methodology and empirical results. Communications of the ACM 26:265-283.

Rosson, Mary Beth
1983 Patterns of experience in text editing. CHI-83. Pp. 171-175 in Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery.
1984 Effects of experience on learning, using, and evaluating a text editor. Human Factors 26:463-475.

Sheppard, S.B., and Kruesi, E.
1981 The effects of the symbology and spatial arrangement of software specifications in a coding task. General Electric Company Information Systems Programs Report TR-81-388200-3. Arlington, Va.: General Electric.

Shneiderman, B.
1982 Systems message design: guidelines and experimental results. In A. Badre and B. Scheiderman, eds., Directions in Human-Computer Interaction. Norwood, N.H.: Ablex.

Smith, S.L.
1982 User-system interface design for computer-based information systems. Mitre Corporation Report ESD-TR-82-132. Bedford, Mass.: Mitre Corporation.

Smith, S.L., and Mosier, J.N.
1984 Design guidelines for user-system interface software. Mitre Corporation Report ESD-TR-84-190. Bedford, Mass.: Mitre Corporation.

Stallman, R.M.
1980 EMACS Manual for ITS users. AI Lab Memo 554. Masschusetts Institute of Technology, Cambridge, Mass.

Sullivan, M.A., and Chapanis, A.
1983 Human factoring: a text editor manual. Behaviour and Information Technology 2:113-125.

Teitelman, W.
1972 Do what I mean: the programmer's assistant. Computers and Automation 21(4)8-11.

Thomas, J.C., and Gould, J.D.
1975 A psychological study of query by example. In Proceedings of the National Computer Conference. Arlington, Va.: American Federation of Information Processing Societies Press.

Van Cott, H., and Kinkade, R.G., eds.
1972 Human Engineering Guide to Equipment Design. Prepared by the American Institutes for Research for the U.S. Department of Defense. Available from the U.S. Government Printing Office, Washington, D.C.: U.S. Department of Defense.

Wasserman, Anthony I.
1982 The user software engineering methodology: an overview. Pp. 591-628 in A.A. Verrijn-Stuart, ed., Information System Design Methodologies. Amsterdam: North Holland Press.

Wasserman, Anthony I., and Shewmake, David T.
1982 Rapid prototyping of interactive information systems. Software Engineering Notes 7(5):171-180.

Whiteside, J., Archer, N., Wixon, D., Good, M.
1982 How do people really use text editors? Pp. 29-40 in Proceedings of the SIGOA Conference on Office Information Systems, Philadelphia.

Wienberg, G.M., and Friedman, D.P.
1984 Reviews, walk-throughs, and inspections. IEEE Transactions on Software Engineering SE-10(1).

Williges, R.C., and Williges, B.H.
1982 Modeling the human operator in computer based data entry. Human Factors 24:285-299.

Wixon, D.R., Whiteside, J.A., Good, M.D., and Jones, J.R.
1983 Building a user defined interface. CHI-83. Pp. 24-27 in Proceedings of the Conference on Human Factors in Computing Systems. New York: Association of Computing Machinery.

Woodson, W.E., and Conover, D.W.
1966 Engineering Guide for Equipment Designers. 2nd ed. Berkeley: University of California Press.

Yunten, T., and Hartson, H.R.
1984 A Supervisory Methodology and Notation (SUPERMAN). In H.R. Hartson, ed, Advances In Human-Computer Interaction. Norwood, N.H.: Ablex.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Methods for Designing Software to Fit Human Needs and Capabilities: Proceedings of the Workshop on Software Human Factors	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Nancy S. Anderson and Judith Reitman Olson, eds.	8. CONTRACT OR GRANT NUMBER(s): N00014-85-G-0093
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Committee on Human Factors, National Research Council, Washington, D. C. 20418	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit No. NR196-167
11. CONTROLLING OFFICE NAME AND ADDRESS: Engineering Psychology Program, Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: May, 1985
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release: Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The Committee on Human Factors is sponsored by the Air Force Office of Scientific Research, the Army Research Institute, the National Aeronautics and Space Administration, the National Science Foundation, and the Office of Naval Research.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

information systems	Committee on Human Factors
computer	software design
software	human factors methods
human-computer interaction	software design methods

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Computer software is usually developed with little or no user input into the design process frequently resulting in sophisticated computer programs that are difficult to use or require extensive training programs. The workshop was convened to identify and describe methods that software designers can employ that incorporate the user into the design team. The result is more effective use and evaluation of the software programs, improved manuals, and reduced training time.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Selected Publications of the Committee on Human Factors

Research Needs for Human Factors (1983)
Research Needs on the Interaction Between Information Systems and Their Users (1984)
Research Issues in Simulator Sickness (1984)
Research and Modeling of Supervisory Control Behavior (1984)